Imagine Applications and Lessons Learned

John Owens
Imagine Application Catalog

• Signal processing
  – Space-time Adaptive Processing (STAP)
    • QR Matrix Decomposition
  – Coherent Side Lobe Cancellation (ISI)
  – Beam Steering (ISI)
  – FFT
  – DCT
  – FIR filters

• Image/Video processing
  – MPEG2 encode
  – Stereo depth extraction

• Graphics
  – Polygon rendering
  – Programmable shading
Sustained Application Performance

- **Stereo Depth Extraction**
  - 320x240 8-bit grayscale
  - 200 frames/second

- **Polygon Rendering**
  - 4.5 Million Vertices/sec
  - 5.1 Million Pixels/sec

- **MPEG Encoding**
  - 720x486 24-bit color
  - 120 frames/second

SPECviewperf ADVS benchmark (unlit)
FIR Filtering

- 16-bit fixed-point
- 13 taps, 2048 outputs

\[ y(n) = \sum_{i=0}^{M-1} k_i x(n-i) \]

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Frequency (MHz)</th>
<th>Performance (GOPS)</th>
<th>Global RF BW (GB/s)</th>
<th>Ops per Byte of GRF BW</th>
<th>Local RF BW (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UltraSparc II</td>
<td>450</td>
<td>0.14</td>
<td>10.63</td>
<td>0.013</td>
<td>---</td>
</tr>
<tr>
<td>Pentium 3 MMX</td>
<td>500</td>
<td>1.64</td>
<td>18.18</td>
<td>0.090</td>
<td>---</td>
</tr>
<tr>
<td>C62</td>
<td>250</td>
<td>0.84</td>
<td>8.49</td>
<td>0.099</td>
<td>---</td>
</tr>
<tr>
<td>Vector</td>
<td>500</td>
<td>7.71</td>
<td>105.06</td>
<td>0.073</td>
<td>---</td>
</tr>
<tr>
<td>Imagine</td>
<td>500</td>
<td>16.11</td>
<td>4.77</td>
<td>3.375</td>
<td>294.15</td>
</tr>
</tbody>
</table>
QRD Performance

![QRD Performance Chart](image-url)
Raw Performance

16-bit applications

floating-point application

23.9

25.6

16-bit kernels

floating-point kernel

depth
mpeg
qrd
dct
convolve
fft

GOPS

12.1
19.8
11.0
23.9
25.6
7.0
### Data Parallelism in Clusters

<table>
<thead>
<tr>
<th>Kernel</th>
<th>1 to 8 Cluster Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT (1024)</td>
<td>6.4</td>
</tr>
<tr>
<td>DCT (8x8)</td>
<td>7.8</td>
</tr>
<tr>
<td>Blockwarp (8x8)</td>
<td>7.2</td>
</tr>
<tr>
<td>Transform (?)</td>
<td>8.0</td>
</tr>
<tr>
<td>Harmonic Mean</td>
<td>7.3</td>
</tr>
</tbody>
</table>
Short Stream Effects

- Processing short streams incurs several penalties:
  - Most kernels have prologue/epilogue basic blocks
  - SWP on kernels leads to setup/teardown of SWP loop
  - Running each kernel takes 200-500 Imagine cycles on host

- Example: ADVS-1 vs. ADVS-8
  - ADVS-8 has more complex texture accesses than ADVS-1
    - ADVS-1 has larger batch size (224 vs. 48)
  - Both have identical geometry
    - ADVS-1: 626k cycles
    - ADVS-8: 912k cycles
Reflections on polygon rendering

- Short stream effects hurt overall performance
- Software pipelining is difficult with varying input sizes
- Continuum of conditional support:
  - Fine: use `select(a, b, c) == a ? b : c`
  - Coarse: divide heterogeneous stream into several homogeneous streams, process homogeneous
  - How big is the gap between these two?
    - If small, MIMD not useful
- Overflow: Very difficult to handle currently
- Ordering: Problematic
General Lessons

- Write a functional simulator first
  - Refine language by writing real applications
- Use a C++ subset if possible
  - Use C++ debugger
- Start with an existing compiler framework
- Good visualization tools are very helpful
  - Debugging of tools
  - Optimization of applications
StreamC Lesson: Use specialization

• Many apps are very predictable given a few fixed parameters
  – E.g. MPEG encoding given image size
• Specialization based on these parameters limits control flow, defines stream sizes
• Good SRF allocation needs fixed (or limited) stream sizes
StreamC Lesson: No streams-by-reference

- Two potential kinds of streams:
  - Simple array of records
    - E.g. triangles
  - Records in another data structure
    - E.g. first row of matrix

- Second kind should copy data into stream, not treat stream as reference to multiple records
  - cleaner syntax
  - simpler compiler
  - compiler can still eliminate copies if possible
StreamC Lesson: Essential optimizations

• Stripmining
  – Process initial input in batches

• Software pipelining
  – Kernels in one stage can cover memory access time in another stage
StreamC Lesson: Variable-size streams

- Variable size streams need special support
- Allocate space in SRF for common case
- When space is filled either:
  - Double-buffer to memory
  - Suspend kernel and process output (harder)
KernelC Lessons

• Never expose the hardware in the language
  – E.g. no explicit communication between SIMD clusters

• Optimizations that are too expensive for normal programs are good for kernels
  – E.g. try multiple schedules w/ some random variation
My wish list for hardware

• **Stream hardware**
  – Fully on-chip stream controller (no need for host component)
    • More sophisticated interaction possible
      – Partial double buffering
    • Better handling of stream conditionals (if a then kernel1 else kernel2)

• **Clusters**
  – More conditional stream bandwidth into/out of clusters
    • Ideal: Equal to unconditional stream bandwidth
  – Reciprocal square root
  – Efficient broadcast?
    • 1 word/cycle currently

• **Scalar unit**
  – e.g. Huffman encode
My wish list for software

• Kernel scheduler:
  – Existing compiler infrastructure for front end
    • Writing inline function support has been painful
  – Scheduling is slow
  – Large kernels overwhelm kernel scheduler’s register allocator
  – Language features:
    • Auto-communication: Ideally, kernels should not know the number of clusters
      – Polygon rendering: only sort/merge know the number of clusters
    • Collapse stream into single scalar value
      – e.g. tree sum

• Stream scheduler:
  – Compiler not profiler
  – Finding proper stripmine batch size is tedious
  – Automate SWP
Future stream hardware

• 1-D streams are great. How about 2-D, 3-D?
  – Concept of “neighborhood” in a stream

• Ordering constraints?